Particle swarm optimization applied to job scheduling problem

Karel Hybner

September 11, 2018

1 Introduction

In this paper we implement the MPSO algorithm developed in [Chen et al., 2013] and evaluate its performance using MATLAB software. The performance of the algorithm is evaluated using classical job scheduling problem benchmarks from OR library [Beasley, 1990]. Influence of different choice of algorithm parameters on its performance is briefly discussed.

1.1 Job-shop Scheduling Problem

Job-shop Scheduling Problem (JSSP) is a combinatorial NP-hard optimization problem. Consider n jobs and m machines, where each of n jobs must operate once on each of m machines in set sequence in order to be completed. The sequence of operations is different for each job. Each *operation* o_{ij} of i-th job on j-th machine lasts a given time t_{ij}. No more than one operation can be processed on one machine at any given time. The optimization objective is then to find a schedule of $n \cdot m$ operations that minimizes the completion time of n jobs, i.e. completion time of the last operation, called *makespan*.

1.2 Particle Swarm Optimization

Particle Swarm Optimization (PSO) is a heuristic technique inspired by behaviour of a flock of birds searching for food. It solves an optimization problem by having a population of candidate solutions (*swarm of particles*) and moving these around in the given searching space. Each movement of these particles is affected by particle's respective best position (position minimizing the value of objective function of the given problem) as well as by the best positions of other particles (i.e. ,in a basic case, by the global best position). The standard PSO algorithm goes as follows:

Algorithm 1 Basic PSO algorithm

Initialize the swarm of N particles with random positions p_i and random velocities v_i; $i = 1, \ldots, N$.
while stopping condition is not met **do**
  Evaluate the fitness of each paricle by evaluating objective function of the problem.
  for $i = 1 \ldots, N$ **do**
    Update the personal best position p_i^{best} of i-th particle and the global best position p_g^{best}.
    Generate U_1, $U_2 \sim \mathcal{U}(0, 1)$,
    Update the velocity of $i - th$ particle using the following rule:

$$v_i^{new} = \omega \cdot v_i + C_1 \cdot U_1 \cdot \left(p_i^{best} - p_i\right) + C_2 \cdot U_2 \cdot \left(p_g^{best} - p_i\right). \tag{1}$$

    Update the position of i-th particle using:

$$p_i^{new} = p_i + v_i^{new}. \tag{2}$$

  end for
end while

2 Application of PSO on job scheduling problem

2.1 Random key encoding scheme

Original PSO was developed to solve continous problems. To use particle swarm optimization for job scheduling problem, a suitable representation of the discrete job schedule in the continous space needs to be used. One of the options is to use random key encoding scheme. In that case, the searching space becomes a continous $n \cdot m$ dimensional space, where the position of each particle consists of $n \cdot m$ real numbers. To get the schedule of operations, this real vector has to be transformed by the following RK scheme:

1. Rank the real vector in an ascending order.
2. With the new vector $\mathbf{r}$ obtained, compute elementwise $(\mathbf{r} \bmod n) + 1$.
3. Resulting vector consists of integers $1, \ldots, n$ and thus directly represents the job schedule.

An example ($n = 3$, $m = 2$):

$$(1.3, 2.4, 3.5, 0.7, 1.6, 3.1) \rightarrow (2, 4, 6, 1, 3, 5) \rightarrow (3, 2, 1, 2, 1, 3) \leftrightarrow (o_{31}, o_{21}, o_{11}, o_{22}, o_{12}, o_{32}).$$

2.2 Individual enhancement scheme using simulated annealing

To improve the performance of the PSO algorithm by enhancing its local search ability, [Chen et al., 2013] proposed a multiple-type individual enhancement scheme consisting of operations swap, insertion, inversion and jump. The procedure of each of these operations starts with randomly chosing dimension p, q (and r in case of a jump). An example is given to explain the difference between respective operations.

- Swap; $p = 1,\ q = 3$

$$(\mathbf{1.3}, 2.4, \mathbf{3.5}, 0.7, 1.6, 3.1) \rightarrow (\mathbf{3.5}, 2.4, \mathbf{1.3}, 0.7, 1.6, 3.1)$$

- Insertion; $p = 1,\ q = 3$

$$(\mathbf{1.3}, 2.4, 3.5, 0.7, 1.6, 3.1) \rightarrow (2.4, 3.5, \mathbf{1.3}, 0.7, 1.6, 3.1)$$

- Inversion; $p = 1,\ q = 3$

$$(\mathbf{1.3}, \mathbf{2.4}, \mathbf{3.5}, 0.7, 1.6, 3.1) \rightarrow (\mathbf{3.5}, \mathbf{2.4}, \mathbf{1.3}, 0.7, 1.6, 3.1)$$

- Jump; $p = 1,\ q = 3,\ r = 5$

$$(\mathbf{1.3}, \mathbf{2.4}, \mathbf{3.5}, 0.7, 1.6, 3.1) \rightarrow (\mathbf{3.5}, 0.7, 1.6, 3.1, \mathbf{1.3}, \mathbf{2.4})$$

Combining this scheme with simmulated annealing procedure results in multi-type individual enhancement scheme (algorithm 2). By inserting it into general PSO framework we finally get MPSO (algorithm 3) as presented in [Chen et al., 2013].

Algorithm 2 Multi-type individual enhancement scheme

Input: P, the individual (particle) to be enhanced; T, starting temperature of SA; T_f, final temperature of SA; cooling rate β

Output: an enhanced particle

while $(T > T_f)$, i.e. while a starting temperature of SA (T) is higher than a final temperature T_f **do**
  Randomly select enhancing operation and generate new individual (particle) P'.
  Evaluate the objective funtion (i.e. evaluate the makespan of an operation sequence represented by P'
  Compute Δ=makespan(P')-makespan(P), where P is a particle before enhancement.
  if $\Delta > 0$ **then**
    randomly generate $R \sim \mathcal{U}(0,1)$
    if $R < \exp\left(\frac{-\Delta}{T}\right)$ **then**
      $P \leftarrow P'$; makespan(P)$\leftarrow$ makespan(P') i.e. accept the worst state with probability $\exp\left(\frac{-\Delta}{T}\right)$
    end if
  else
    $P \leftarrow P'$; makespan(P) $\leftarrow$ makespan(P'); $T \leftarrow \beta \cdot T$
  end if
end while

Algorithm 3 MPSO algorithm

Input: P_{enh}, probability of executing enhancement scheme; PSO parameters C_1, C_2, ω; $MaxEval$, maximum number of makespan evaluations or, alternatively, $MaxIter$, maximum number of iterations

Output: the best job schedule represented by p_g^{best}

Initialize the positions and velocities of each of N particles
while stopping condition (e.g. MaxEval or MaxIter) is not reached **do**
  for $i = 1 \ldots, N$ **do**
    Generate $S \sim \mathcal{U}(0,1)$
    if $S < P_{enh}$ **then**
      Execute the individual enhancement scheme for i-th particle (alg. 2)
    end if
    Update the local best position of i-th particle
  end for
  Update the global best position p_g^{best}.
  for $i = 1 \ldots, N$ **do**
    Update the velocity and position of i-th particle.
  end for
end while

3 Experimental results

We use five test benchmarks from OR-library, specifically instances FT06, FT10, LA01, LA02 and LA24 presented first in [Fisher and Thompson, 1963] and [Lawrence, 1984] to test the performance of implemented algorithms. The parameters of MPSO and PSO are set to the values specified in table 3. Particles of the swarm are initialized randomly.

In the first experiment, both MPSO and PSO are tested. Five runs for each of the data instances are executed. The results are shown in table 1. BKS denotes here the best known solution of respective instances.

N	C_1	C_2	ω	P_{enh}	MaxEval	T	T_f	β
30	1.5	2	$0.99^{\#iterations}$	0.01	$3 \cdot 10^5$	makespan(P)-BKS	0.1	0.99

Table 1: MPSO & PSO setting

		MPSO			PSO		
	BKS	Best	Avg	Worst	Best	Avg	Worst
FT06	55	55	55	55	55	55	55
FT10	930	976	995.6	1013	1033	1092.2	1152
LA01	666	666	666	666	666	671.4	693
LA02	655	655	655.8	657	669	695	732
LA24	935	1014	1017.6	1021	1103	1192.2	1268

Table 2: Results of 5 runs of MPSO & PSO algorithms on benchmark data

As evidenced by the results (tab. 2), MPSO reaches generally better solution in the same number of makespan evaluations. The performance can be further evaluated using the metric of MNE, mean number of evaluations (while reaching the best known solution), and reliability, i.e. number of runs when the BKS has been reached divided by total number of runs (5). We compare only the instances where both algorithms reached BKS at least once. Mean number of evaluations needed to reach the best solution is very significantly lower for MPSO. See table 3.

	MPSO		MPSO	
	MNE	Reliability	MNE	Reliability
FT06	4495.8	1	53970	1
LA01	8332.2	1	92310	0.8

Table 3: Comparison of MNE and reliability for MPSO & PSO algorithms

We compare the results obtained by PSO and MPSO algorithms with results obtained by the Shoot&Go method. The results of five runs of this method on the same dataset are provided in tables 4 and 5. The value of a parameter h denotes here the maximum number of local searches in each iteration, zero being equivalent to random shooting. Maximal number of evaluations is set on the previously used value: $3 \cdot 10^5$. In comparison to both PSO and MPSO algorithms, the performance of Shoot&Go is significantly worse with same number of makespan evaluations.

		Shoot&Go ($h = 0$)			Shoot&Go ($h = 2$)			Shoot&Go ($h = 5$)		
	BKS	Best	Avg	Worst	Best	Avg	Worst	Best	Avg	Worst
FT06	55	55	55	55	55	55.6	57	55	55.2	56
FT10	930	1182	1202.6	1227	1160	1195.6	1190	55	1210.4	1225
LA01	666	678	690.8	700	685	691.2	705	680	684.4	694
LA02	655	718	735.8	752	734	740.4	746	723	738.6	755
LA24	935	1259	1279.6	1302	1267	1280.4	1293	1250	1272.8	1294

Table 4: Results of 5 runs of Shoot&Go algorithm on benchmark data

	Shoot&Go ($h = 0$)		Shoot&Go ($h = 2$)		Shoot&Go ($h = 5$)	
	MNE	Reliability	MNE	Reliability	MNE	Reliability
FT06	210781	1	64935	0.6	170937	0.8
LA01	$\sim$	0	$\sim$	0	$\sim$	0

Table 5: MNE and reliability for Shoot& algorithm

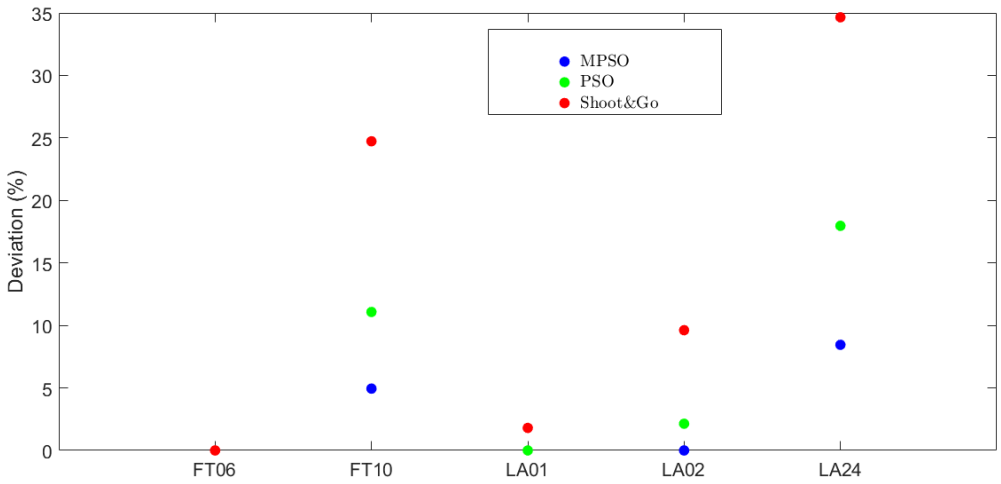

Figure 1: Percentage of the deviation of the best found solution (Best) and the best known solution (BKS)

Choice of parameters

In the second experiment, we vary the number of particles as well as the cooling rate β. Values of other parameters remain the same. For this experiment, only the FT10 dataset is used. Results (tab. 6) are obtained by running the MPSO algorithm five times. The cooling rate parameter β appears to be significant for performance of the algorithm. On the contrary, no influence of swarm size on the results can be derived from results. The results averaged over different values of N and β respectively are presented in figure 2.

$N \backslash \beta$	0.8	0.85	0.9	0.95	0.99
5	1067.6	1027.2	1054.8	1012.6	980.8
10	1075.2	1084.4	1048.2	1028.2	992
20	1070.2	1075.6	1065.8	1027.2	989
30	1098.4	1075.8	1065.2	1032.6	983.2
50	1079.4	1082.6	1060	1035.8	978

Table 6: MPSO run on FT10 with different setting of parameters

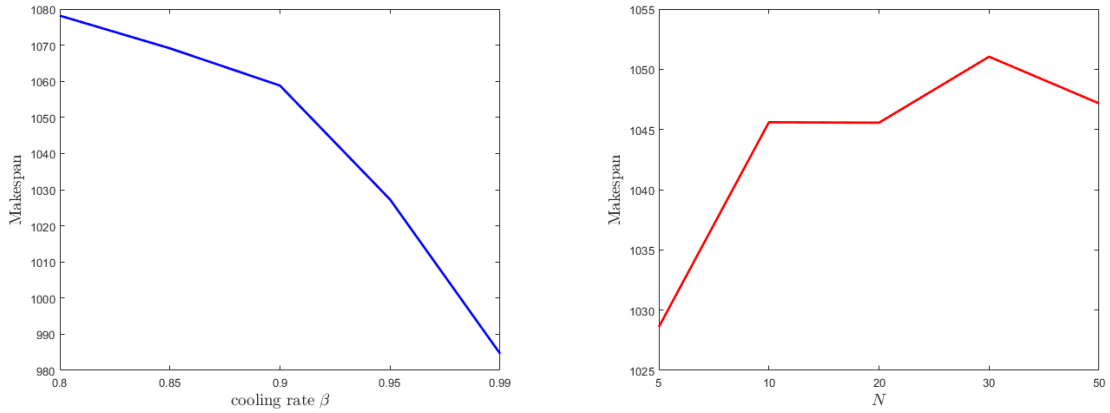

Figure 2: The results of exp. 2 (tab. 4) averaged over different values of N and β respectively

References

[Beasley, 1990] Beasley, J. E. (1990). OR-library: Distributing test problems by electronic mail. *The Journal of the Operational Research Society*, 41(11):1069.

[Chen et al., 2013] Chen, R.-J., Lin, T.-L., Run, R.-S., Kuo, I.-H., Chen, Y.-H., Lai, J.-L., tzong wann, K., and Horng, S.-J. (2013). An efficient job-shop scheduling algorithm based on particle swarm optimization.

[Fisher and Thompson, 1963] Fisher, H. and Thompson, G. (1963). Probabilistic learning combinations of local job-shop scheduling rules. *Journal of Software Engineering and Applications*, pages 225–251.

[Lawrence, 1984] Lawrence, S. (1984). An experimental investigation of heuristic scheduling techniques.in supplement to resource constrained project scheduling.